Data Analysis Report: Weather Data Set

Objective: This report presents the findings of our analysis of weather within our subscription-based service. The goal is to identify key factors contributing to weather and provide actionable recommendations to mitigate it

Import the Required Library:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

The importing of libraries will help data analysts to do analysis on the datasets and get insights about the data.

Load the CSV File:

Use the **read_csv** function to load the CSV file into a Data Frame. Provide the path or URL of the CSV file as an argument to the function.

```
data=pd.read_csv(r'C:\Users\priya\Downloads\data set with questions\data set with questions\Weather Data.csv')
```

This line of code reads the CSV file named `Weather data.csv` and stores its contents in the Data Frame '**data**'.

Exploring the data:

Once loaded the data into Data Frame, you can start exploring it. Common DataFrame operations include:

```
data.head()
```

	Date/Time	Temp_C	Dew Point Temp_C	Rel Hum_%	Wind Speed_km/h	Visibility_km	Press_kPa	Weather
0	01-01-2012 00:00	-1.3	-3.5	18	9	25.0	98.67	Clear
1	01-01-2012 01:00	7.4	2.8	20	24	24.1	99.37	Rain
2	01-01-2012 02:00	15.7	13.4	21	26	25.0	99.84	Cloudy
3	01-01-2012 03:00	4.9	-2.6	27	15	24.1	100.94	Mainly Clear
4	01-01-2012 04:00	-13.4	-19.7	30	4	25.0	102.32	Mostly Cloudy

data.head( ) : Displays the first 5 rows of the DataFrame to get a quick look at the data.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8784 entries, 0 to 8783
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Date/Time         8784 non-null   object 
 1   Temp_C            8784 non-null   float64
 2   Dew Point Temp_C  8784 non-null   float64
 3   Rel Hum_%         8784 non-null   int64  
 4   Wind Speed_km/h   8784 non-null   int64  
 5   Visibility_km     8784 non-null   float64
 6   Press_kPa         8784 non-null   float64
 7   Weather           8784 non-null   object 
dtypes: float64(4), int64(2), object(2)
memory usage: 549.1+ KB
```

data.info() : Show info about the DataFrame, such as data types and missing values.

```
data.describe()
```

	Temp_C	Dew Point Temp_C	Rel Hum_%	Wind Speed_km/h	Visibility_km	Press_kPa
count	8784.000000	8784.000000	8784.000000	8784.000000	8784.000000	8784.000000
mean	8.798144	2.555294	67.431694	14.945469	27.664447	101.051623
std	11.687883	10.883072	16.918881	8.688696	12.622688	0.844005
min	-23.300000	-28.500000	18.000000	0.000000	0.200000	97.520000
25%	0.100000	-5.900000	56.000000	9.000000	24.100000	100.560000
50%	9.300000	3.300000	68.000000	13.000000	25.000000	101.070000
75%	18.800000	11.800000	81.000000	20.000000	25.000000	101.590000
max	33.000000	24.400000	100.000000	83.000000	48.300000	103.650000

data.describe(): Provide summary statistics for numeric columns.

```
data.columns
```

```
Index(['Date/Time', 'Temp_C', 'Dew Point Temp_C', 'Rel Hum_%',
       'Wind Speed_km/h', 'Visibility_km', 'Press_kPa', 'Weather'],
      dtype='object')
```

data.columns(): List the column names.

Analysis Methods:

1. Find all the unique 'Wind Speed' values in the data ¶

```
data['Wind Speed_km/h'].nunique()
```

```
34
```

```
data['Wind Speed_km/h'].unique()
```

```
array([ 9, 24, 26, 15,  4,  0, 19, 17, 11, 22, 35, 13, 20,  6,  7, 30, 32,
       41, 39, 28, 44, 33, 37, 52, 46,  2, 50, 48, 57, 63, 43, 83, 70, 54],
      dtype=int64)
```

There are **34 unique wind speed** values in the data

data['Wind Speed_km/h']: This part of the code selects the 'Wind Speed_km/h' column from the DataFrame 'data'. It accesses the data in the specific column.

.nunique() : This is a Pandas function that operates on a Series (a single column of data) and calculates the number of unique values in that Series. In this case, it calculates the number of unique wind speed values in the 'Wind Speed_km/h' column.

So, when you execute **data['Wind Speed_km/h'].nunique()**, it will return an integer value representing the count of unique wind speed values in the specified column. This can be useful to understand the diversity of wind speeds in the dataset and get a sense of the variability in the 'Wind Speed_km/h' data.

.unique(): This is a Pandas function that operates on a Series (a single column of data) and returns an array containing all the unique values found in that Series. In this case, it retrieves an array of unique wind speed values from the 'Wind Speed_km/h' column.

So, when you execute **data['Wind Speed_km/h'].unique(),** you will get an array of all the distinct wind speed values present in the specified column. This can be useful for various purposes, such as identifying the range of wind speeds in your dataset, checking for anomalies or outliers, or creating a list of unique wind speeds for further analysis or visualization.

2. Find the number of times when the 'Weather is exactly Clear'.

```
data.loc[data['Weather']=='Clear','Weather'].value_counts()
```

```
Clear    1326
Name: Weather, dtype: int64
```

1326 times the **weather** is exactly **clear**

data.loc[]: The **.loc** function is used to locate rows in the DataFrame based on a specified condition. In this case, it's used to select rows where the 'Weather' column is equal to 'Clear'.

.value_counts(): This is a Pandas function that operates on a Series and counts the number of occurrences of each unique value in the Series. In this case, it counts how many times 'Clear' appears in the 'Weather' column after filtering for rows with 'Clear' weather conditions.

data.loc[data['Weather'] == 'Clear','Weather'].value_counts(): It will get a count of how many times the weather condition 'Clear' appears in the 'Weather' column of the DataFrame 'data.' This can be useful for understanding how frequently clear weather occurs in the dataset or for summarizing the frequency of specific categories within a categorical variable like 'Weather'.

3. Find the number of times when the 'Wind Speed was exactly 4 km/h'.

```
data.loc[data['Wind Speed_km/h']==4,'Wind Speed_km/h'].value_counts()
```

```
4    474
Name: Wind Speed_km/h, dtype: int64
```

474 times the **wind speed** was exactly **4 km/h**

data.loc[data['Wind Speed_km/h']==4, 'Wind Speed_km/h']: This part of the code selects the subset of the DataFrame 'data' where the 'Wind Speed_km/h' column contains the value 4. It specifically selects the 'Wind Speed_km/h' column for these rows.

So, when **data.loc[data['Wind Speed_km/h']==4,'Wind Speed_km/h'].value_counts()**, is executed, will get a count of how many times the wind speed is 4 km/h in the 'Wind Speed_km/h' column of the DataFrame 'data.' This can be useful for understanding how frequently wind speeds of 4 km/h occur in the dataset or for summarizing the frequency of specific values within a numeric variable like 'Wind Speed_km/h'.

4. Find out all the Null Values in the data.

```
data.isnull().sum()
```

```
Date/Time           0
Temp_C              0
Dew Point Temp_C    0
Rel Hum_%           0
Wind Speed_km/h     0
Visibility_km       0
Press_kPa           0
Weather             0
dtype: int64
```

This data set is having **ZERO** null values

data.isnull(): This part of the code creates a new DataFrame that has the same shape as 'data' but contains Boolean values (True or False) for each cell. Each element in this new DataFrame is True if the corresponding cell in 'data' is a missing value and False if it is not.

.sum(): The. sum() function is then applied to this Boolean DataFrame. When you apply. sum() to a DataFrame of Booleans, it calculates the sum of True values along each column. Since True is treated as 1 and False as 0 when you sum them up, this effectively counts the number of True values in each column, which corresponds to the number of missing values.

So, when you execute **data.isnull().sum()**, you will get a Series that shows the count of missing values for each column in the DataFrame 'data.' This can be useful for identifying which columns have missing data and for assessing the extent of missingness in your dataset.

5. Rename the column name 'Weather' of the dataframe to 'Weather Condition'.

```
dc=data.rename(columns={'Weather': 'Weather Condition'})
```

```
dc.columns
```

```
Index(['Date/Time', 'Temp_C', 'Dew Point Temp_C', 'Rel Hum_%',
       'Wind Speed_km/h', 'Visibility_km', 'Press_kPa', 'Weather Condition'],
      dtype='object')
```

Changed column name from **Weather-to-Weather Condition**

.rename(columns={'Weather': 'Weather Condition'}): This is a Pandas function that allows you to rename the columns of a DataFrame. In this case:

columns = {'Weather': 'Weather Condition'} is a dictionary where the key **'Weather'** represents the current column name that you want to change, and the value **'Weather Condition'** represents the new column name you want to assign.

6. What is the mean 'Visibility' ?

```
data['Visibility_km'].mean()
```

```
27.664446721311478
```

The **mean** of the visibility is **`27.664446721311478`**

The code **data['Visibility_km'].mean()** is used to calculate the mean (average) value of the **'Visibility_km'** column in a DataFrame called 'data'.

7. What is the Standard Deviation of 'Pressure' in this data?

```
data['Press_kPa'].std()
```

```
0.8440047459486459
```

The **standard deviation** of pressure is **`0.8440047459486459`**

The code **data['Press_kPa'].std()** is used to calculate the standard deviation value of the **'Press_kPa'** column in the DataFrame called 'data'.

8. What is the Variance of 'Relative Humidity' in this data ?

```
data['Rel Hum_%'].var()
```

```
286.2485501985015
```

The **Variance** of pressure is **`286.2485501985015`**

The code **data['Rel Hum_%'].var()** is used to calculate the variance value of the **'Rel Hum_%'** column in the DataFrame called 'data'.

9. Find all instances when 'Snow' was recorded.

```
data.loc[data['Weather']=='Snow',['Date/Time','Weather']]
```

	Date/Time	Weather
11	01-01-2012 11:00	Snow
70	03-01-2012 22:00	Snow
73	04-01-2012 01:00	Snow
105	05-01-2012 09:00	Snow
112	05-01-2012 16:00	Snow
...	...	...
8573	9/22/2012 13:00	Snow
8650	9/25/2012 18:00	Snow
8671	9/26/2012 15:00	Snow
8713	9/28/2012 1:00	Snow
8734	9/28/2012 8:00	Snow

390 rows × 2 columns

Out of **8784** rows there are **390** rows having **snow** weather

The code **data.loc[data['Weather'] == 'Snow', ['Date/Time', 'Weather']]** is used to filter and select specific rows and columns from a DataFrame called 'data' based on a condition.

['Date/Time', 'Weather']: Inside the .loc function, you specify the columns you want to select for the rows that meet the condition. In this case, you are selecting the 'Date/Time' and 'Weather' columns.

10. Find all instances when 'Wind Speed is above 24' and 'Visibility is 25'.

```
data.loc[(data['Wind Speed_km/h']>24) & (data['Visibility_km']==25),['Wind Speed_km/h','Visibility_km']]
```

	Wind Speed_km/h	Visibility_km
2	26	25.0
73	35	25.0
126	39	25.0
158	26	25.0
184	44	25.0
...	...	...
8707	33	25.0
8714	26	25.0
8738	28	25.0
8745	28	25.0
8776	43	25.0

308 rows × 2 columns

Out of 8784 rows of different data there are **308** rows with **wind speed is greater than 24** and **visibility is equal to 25.**

data.loc[] is used to select rows and columns from a Pandas DataFrame called data.

(data['Wind Speed_km/h'] > 24) is a Boolean condition that checks whether the values in the 'Wind Speed_km/h' column of the DataFrame are greater than 24. This condition will create a Boolean Series with True for rows where the wind speed is greater than 24 and False otherwise.

(data['Visibility_km'] == 25) is another Boolean condition that checks whether the values in the 'Visibility_km' column of the DataFrame are equal to 25. This condition will create a Boolean Series with True for rows where the visibility is exactly 25 kilometers and False otherwise.

(data['Wind Speed_km/h'] > 24) & (data['Visibility_km'] == 25) combines the two conditions using the & operator. This results in a Boolean Series that is True only for rows where both conditions are satisfied, meaning the wind speed is greater than 24 km/h, and the visibility is 25 kilometers.

['Wind Speed_km/h', 'Visibility_km'] is a list of column names that you want to select from the DataFrame. In this case, you want to select the 'Wind Speed_km/h' and 'Visibility_km' columns for the rows that meet the combined condition.

11. What is the Mean value of each column against each 'Weather Condition ?

```
g=data.groupby('Weather')
g.mean(numeric_only=True)
```

Weather	Temp_C	Dew Point Temp_C	Rel Hum_%	Wind Speed_km/h	Visibility_km	Press_kPa
Clear	6.825716	0.089367	67.127451	10.557315	30.153243	101.084495
Cloudy	7.970544	2.375810	67.349537	16.127315	26.625752	101.056852
Drizzle	7.353659	5.504878	69.048780	16.097561	17.931707	101.099268
Drizzle,Fog	8.067500	7.033750	70.062500	11.862500	5.257500	100.820750
Drizzle,Ice Pellets,Fog	0.400000	-0.700000	52.000000	20.000000	4.000000	99.440000
Drizzle,Snow	1.050000	0.150000	44.000000	14.000000	10.500000	100.490000
Drizzle,Snow,Fog	0.693333	0.120000	69.800000	15.533333	5.513333	100.971333
Fog	4.303333	3.159333	66.466667	7.946667	6.248000	101.149400
Freezing Drizzle	-5.657143	-8.000000	68.857143	16.571429	9.200000	101.070000
Freezing Drizzle,Fog	-2.533333	-4.183333	64.000000	17.000000	5.266667	100.851667
Freezing Drizzle,Haze	-5.433333	-8.000000	63.333333	10.333333	2.666667	101.136667
Freezing Drizzle,Snow	-5.109091	-7.072727	62.454545	16.272727	5.872727	100.380909
Freezing Fog	-7.575000	-9.250000	68.000000	4.750000	0.650000	101.222500
Freezing Rain	-3.885714	-6.078571	60.785714	19.214286	8.242857	101.500714
Freezing Rain,Fog	-2.225000	-3.750000	52.750000	15.500000	7.550000	100.267500
Freezing Rain,Haze	-4.900000	-7.450000	63.000000	7.500000	2.400000	100.265000
Freezing Rain,Ice Pellets,Fog	-2.600000	-3.700000	65.000000	28.000000	8.000000	98.330000
Freezing Rain,Snow Grains	-5.000000	-7.300000	92.000000	32.000000	4.800000	102.520000
Haze	-0.200000	-2.975000	69.625000	10.437500	7.831250	100.805625
Mainly Clear	12.558927	4.581671	68.020893	14.144824	34.264862	101.040940
Moderate Rain,Fog	1.700000	0.800000	89.000000	17.000000	6.400000	100.450000
Moderate Snow	-5.525000	-7.250000	67.500000	33.750000	0.750000	100.760000
Moderate Snow,Blowing Snow	-5.450000	-6.500000	81.500000	40.000000	0.600000	102.215000
Mostly Cloudy	10.574287	3.131174	67.214113	15.813920	31.253842	101.051054
Rain	9.786275	7.042810	67.614379	19.254902	18.856536	101.051797
Rain Showers	13.722340	9.187766	68.335106	17.132979	22.816489	101.020106
Rain Showers,Fog	12.800000	12.100000	31.000000	13.000000	6.400000	99.800000
Rain Showers,Snow Showers	2.150000	-1.500000	68.500000	22.500000	21.700000	101.080000
Rain,Fog	8.273276	7.219828	66.818966	14.793103	6.873276	100.991983
Rain,Haze	4.633333	2.066667	57.666667	11.666667	6.700000	100.716667
Rain,Ice Pellets	0.600000	-0.600000	54.000000	24.000000	9.700000	101.880000
Rain,Snow	1.055556	-0.566667	66.944444	28.388889	11.672222	100.895000
Rain,Snow Grains	1.900000	-2.100000	87.000000	26.000000	25.000000	100.870000
Rain,Snow,Fog	0.800000	0.300000	61.000000	9.000000	6.400000	102.480000
Rain,Snow,Ice Pellets	1.100000	-0.175000	72.500000	23.250000	6.000000	101.170000
Snow	-4.524103	-7.623333	66.402564	20.038462	11.171795	101.077205
Snow Pellets	0.700000	-6.400000	66.000000	35.000000	2.400000	99.560000
Snow Showers	-3.506667	-7.866667	65.600000	19.233333	20.158333	100.999333

Snow Showers,Fog	-10.675000	-11.900000	63.750000	13.750000	7.025000	100.770000
Snow,Blowing Snow	-5.410526	-7.621053	72.631579	34.842105	4.105263	101.032105
Snow,Fog	-5.075676	-6.364865	70.459459	17.324324	4.537838	101.194865
Snow,Haze	-4.020000	-6.860000	66.000000	5.000000	4.640000	100.360000
Snow,Ice Pellets	-1.883333	-3.666667	74.000000	23.833333	7.416667	100.746667
Thunderstorms	24.150000	19.750000	56.500000	7.500000	24.550000	101.375000
Thunderstorms,Heavy Rain Showers	10.900000	9.000000	82.000000	9.000000	2.400000	101.400000
Thunderstorms,Moderate Rain Showers,Fog	19.600000	18.500000	58.000000	15.000000	3.200000	99.940000
Thunderstorms,Rain	20.433333	18.533333	71.666667	15.666667	19.833333	101.536667
Thunderstorms,Rain Showers	20.037500	17.618750	68.437500	18.312500	15.893750	100.976875
Thunderstorms,Rain Showers,Fog	21.600000	18.700000	58.666667	19.666667	9.700000	100.806667
Thunderstorms,Rain,Fog	20.600000	18.600000	42.000000	19.000000	4.800000	100.450000

The following data frame shows the average of each column against each weather condition.

For **clear weather condition** the average Temperature is **6.825716.**

For **cloudy weather** the average temperature is 7.970544. and so on….

g.mean(numeric_only=True): This part of the code calculates the mean (average) for each group within the GroupBy object 'g.'

The **numeric_only=True** argument is used to ensure that only numeric columns are included in the calculation. If there are non-numeric columns in the DataFrame, they will be excluded from the mean calculation.

The result will be a new DataFrame where the rows correspond to unique values in the 'Weather' column, and the columns represent the mean values of numeric columns for each weather category.

For example, if your original 'data' DataFrame had columns like 'Temperature,' 'Humidity,' and 'Wind_Speed,' and you used this code to group by 'Weather,' the resulting DataFrame would have rows corresponding to different weather conditions (e.g., 'Sunny,' 'Rainy,' 'Cloudy') and columns representing the mean values of 'Temperature,' 'Humidity,' and 'Wind_Speed' for each weather condition.

12. What is the Minimum & Maximum value of each column against each 'Weather Condition ?

```
g.min()
```

Weather	Date/Time	Temp_C	Dew Point Temp_C	Rel Hum_%	Wind Speed_km/h	Visibility_km	Press_kPa
Clear	01-01-2012 00:00	-23.3	-28.5	18	0	11.3	97.75
Cloudy	01-01-2012 02:00	-21.4	-26.8	20	0	11.3	97.52
Drizzle	01-06-2012 08:00	1.1	-0.2	37	0	6.4	98.29
Drizzle,Fog	01-04-2012 01:00	0.0	-1.6	38	0	1.0	98.32
Drizzle,Ice Pellets,Fog	7/24/2012 5:00	0.4	-0.7	52	20	4.0	99.44
Drizzle,Snow	05-02-2012 09:00	0.9	0.1	39	9	9.7	100.27
Drizzle,Snow,Fog	03-11-2012 20:00	0.3	-0.1	46	7	2.4	99.26
Fog	01-01-2012 13:00	-16.0	-17.2	21	0	0.2	97.97
Freezing Drizzle	04-01-2012 03:00	-9.0	-12.2	43	6	4.8	99.75
Freezing Drizzle,Fog	10/15/2012 4:00	-6.4	-9.0	31	6	3.6	98.81
Freezing Drizzle,Haze	01-06-2012 00:00	-5.8	-8.3	32	9	2.0	100.55
Freezing Drizzle,Snow	02-06-2012 07:00	-8.3	-10.4	37	6	2.4	99.74
Freezing Fog	12/31/2012 2:00	-19.0	-22.9	34	0	0.2	100.66
Freezing Rain	01-03-2012 13:00	-6.5	-9.0	40	7	2.8	100.92
Freezing Rain,Fog	07-05-2012 15:00	-6.1	-8.7	35	7	2.8	99.45
Freezing Rain,Haze	11/23/2012 7:00	-4.9	-7.5	57	6	2.0	100.23
Freezing Rain,Ice Pellets,Fog	8/16/2012 23:00	-2.6	-3.7	65	28	8.0	98.33
Freezing Rain,Snow Grains	04-02-2012 07:00	-5.0	-7.3	92	32	4.8	102.52
Haze	01-02-2012 17:00	-11.5	-16.0	37	0	4.8	99.27
Mainly Clear	01-01-2012 03:00	-22.8	-28.0	20	0	12.9	97.84
Moderate Rain,Fog	8/20/2012 16:00	1.7	0.8	89	17	6.4	100.45
Moderate Snow	10-11-2012 22:00	-6.3	-7.6	29	26	0.6	99.93
Moderate Snow,Blowing Snow	06-07-2012 08:00	-5.5	-6.6	67	39	0.6	101.97
Mostly Cloudy	01-01-2012 04:00	-23.2	-28.5	18	0	11.3	97.56
Rain	01-01-2012 01:00	0.3	-5.7	20	0	4.0	98.06
Rain Showers	01-03-2012 11:00	1.6	-7.2	24	0	6.4	97.93
Rain Showers,Fog	12/17/2012 16:00	12.8	12.1	31	13	6.4	99.80
Rain Showers,Snow Showers	8/31/2012 11:00	2.1	-1.8	67	17	19.3	100.54
Rain,Fog	01-02-2012 12:00	0.0	-1.2	23	0	2.0	98.70
Rain,Haze	01-02-2012 19:00	4.0	1.0	40	7	4.0	99.89
Rain,Ice Pellets	10-03-2012 01:00	0.6	-0.6	54	24	9.7	101.88
Rain,Snow	04-09-2012 20:00	0.6	-1.7	31	13	2.4	100.03
Rain,Snow Grains	07-01-2012 14:00	1.9	-2.1	87	26	25.0	100.87
Rain,Snow,Fog	05-10-2012 03:00	0.8	0.3	61	9	6.4	102.48
Rain,Snow,Ice Pellets	02-09-2012 11:00	0.9	-0.7	53	17	4.8	100.30
Snow	01-01-2012 11:00	-16.7	-24.6	20	0	1.0	97.99
Snow Pellets	7/19/2012 2:00	0.7	-6.4	66	35	2.4	99.56
Snow Showers	01-03-2012 05:00	-13.3	-19.3	31	0	2.4	99.09

Snow Showers,Fog	1/20/2012 3:00	-11.3	-12.7	56	7	4.0	100.33
Snow,Blowing Snow	01-09-2012 10:00	-12.0	-16.2	44	24	0.6	99.23
Snow,Fog	01-02-2012 09:00	-10.1	-12.0	38	4	1.2	99.60
Snow,Haze	01-06-2012 17:00	-4.3	-7.2	48	0	4.0	98.58
Snow,Ice Pellets	01-03-2012 17:00	-4.3	-5.9	50	19	2.8	100.13
Thunderstorms	11/29/2012 16:00	21.6	19.4	56	0	24.1	100.86
Thunderstorms,Heavy Rain Showers	11-05-2012 12:00	10.9	9.0	82	9	2.4	101.40
Thunderstorms,Moderate Rain Showers,Fog	10-01-2012 14:00	19.6	18.5	58	15	3.2	99.94
Thunderstorms,Rain	10/23/2012 4:00	19.4	18.2	64	4	16.1	100.56
Thunderstorms,Rain Showers	01-11-2012 11:00	11.0	7.0	44	7	6.4	99.40
Thunderstorms,Rain Showers,Fog	08-01-2012 15:00	19.5	16.1	34	7	9.7	99.33
Thunderstorms,Rain,Fog	11-04-2012 16:00	20.6	18.6	42	19	4.8	100.45

The above data frame shows the minimum values of each column against all kind of weather conditions.

The column called weather is grouped by categories and stored to a variable called ‘g’, and the aggregate function ‘.min()’ is applied to the grouped data. It’ll return the minimum value of each column against each weather condition.

```
g.max()
```

Weather	Date/Time	Temp_C	Dew Point Temp_C	Rel Hum_%	Wind Speed_km/h	Visibility_km	Press_kPa
Clear	9/30/2012 7:00	32.8	20.4	100	33	48.3	103.63
Cloudy	9/30/2012 8:00	30.5	22.6	100	54	48.3	103.52
Drizzle	9/15/2012 22:00	18.8	17.7	97	30	25.0	103.58
Drizzle,Fog	9/19/2012 15:00	19.9	19.1	98	28	9.7	103.56
Drizzle,Ice Pellets,Fog	7/24/2012 5:00	0.4	-0.7	52	20	4.0	99.44
Drizzle,Snow	3/17/2012 1:00	1.2	0.2	49	19	11.3	100.71
Drizzle,Snow,Fog	9/21/2012 12:00	1.1	0.6	94	32	9.7	102.47
Fog	9/30/2012 19:00	20.8	19.6	99	22	9.7	103.22
Freezing Drizzle	8/21/2012 5:00	-2.3	-3.3	89	26	12.9	101.78
Freezing Drizzle,Fog	7/26/2012 6:00	-0.3	-2.3	80	33	8.0	103.01
Freezing Drizzle,Haze	5/21/2012 4:00	-5.0	-7.7	81	11	4.0	101.83
Freezing Drizzle,Snow	8/18/2012 4:00	-3.3	-4.6	90	24	12.9	101.15
Freezing Fog	5/14/2012 9:00	-0.1	-0.3	86	9	0.8	101.64
Freezing Rain	8/22/2012 10:00	0.3	-1.7	100	28	16.1	102.45
Freezing Rain,Fog	5/16/2012 23:00	0.1	-0.9	77	26	9.7	101.21
Freezing Rain,Haze	3/25/2012 23:00	-4.9	-7.4	69	9	2.8	100.30
Freezing Rain,Ice Pellets,Fog	8/16/2012 23:00	-2.6	-3.7	65	28	8.0	98.33
Freezing Rain,Snow Grains	04-02-2012 07:00	-5.0	-7.3	92	32	4.8	102.52
Haze	9/25/2012 15:00	14.1	11.1	98	17	9.7	103.29
Mainly Clear	9/30/2012 3:00	33.0	21.2	100	63	48.3	103.65
Moderate Rain,Fog	8/20/2012 16:00	1.7	0.8	89	17	6.4	100.45
Moderate Snow	6/17/2012 14:00	-4.9	-6.7	85	39	0.8	101.96

Moderate Snow,Blowing Snow	6/26/2012 2:00	-5.4	-6.4	96	41	0.6	102.46
Mostly Cloudy	9/30/2012 9:00	32.4	24.4	100	83	48.3	103.63
Rain	9/30/2012 0:00	22.8	20.4	97	52	48.3	103.59
Rain Showers	9/29/2012 7:00	26.4	23.0	99	41	48.3	103.65
Rain Showers,Fog	12/17/2012 16:00	12.8	12.1	31	13	6.4	99.80
Rain Showers,Snow Showers	8/31/2012 7:00	2.2	-1.2	70	28	24.1	101.62
Rain,Fog	9/29/2012 11:00	21.7	19.5	93	46	9.7	102.71
Rain,Haze	12-10-2012 11:00	5.5	2.9	75	17	9.7	101.52
Rain,Ice Pellets	10-03-2012 01:00	0.6	-0.6	54	24	9.7	101.88
Rain,Snow	9/26/2012 11:00	1.7	0.5	93	52	25.0	102.21
Rain,Snow Grains	07-01-2012 14:00	1.9	-2.1	87	26	25.0	100.87
Rain,Snow,Fog	05-10-2012 03:00	0.8	0.3	61	9	6.4	102.48
Rain,Snow,Ice Pellets	7/19/2012 22:00	1.3	0.1	86	28	6.4	101.90
Snow	9/28/2012 8:00	3.7	0.3	100	57	25.0	103.65
Snow Pellets	7/19/2012 2:00	0.7	-6.4	66	35	2.4	99.56
Snow Showers	8/31/2012 0:00	2.9	-0.7	95	37	48.3	102.45
Snow Showers,Fog	7/19/2012 13:00	-10.0	-11.1	76	22	9.7	101.48
Snow,Blowing Snow	9/25/2012 9:00	-1.4	-2.9	97	48	9.7	103.59
Snow,Fog	9/22/2012 12:00	1.1	0.8	99	35	9.7	103.51
Snow,Haze	12/13/2012 14:00	-3.6	-6.4	83	15	6.4	101.90
Snow,Ice Pellets	9/23/2012 20:00	0.8	-1.7	92	33	11.3	101.73
Thunderstorms	9/25/2012 5:00	26.7	20.1	57	15	25.0	101.89
Thunderstorms,Heavy Rain Showers	11-05-2012 12:00	10.9	9.0	82	9	2.4	101.40
Thunderstorms,Moderate Rain Showers,Fog	10-01-2012 14:00	19.6	18.5	58	15	3.2	99.94
Thunderstorms,Rain	9/19/2012 14:00	21.3	19.1	80	30	24.1	102.82
Thunderstorms,Rain Showers	9/17/2012 13:00	25.5	23.1	95	32	25.0	102.55
Thunderstorms,Rain Showers,Fog	8/15/2012 9:00	22.9	21.3	82	35	9.7	101.77
Thunderstorms,Rain,Fog	11-04-2012 16:00	20.6	18.6	42	19	4.8	100.45

The above data frame shows the maximum values of each column against each weather condition.

13. Show all the Records where Weather Condition is Fog.

```
pd.DataFrame(data.loc[data['Weather']=='Fog','Weather'])
```

	Weather
13	Fog
53	Fog
136	Fog
197	Fog
278	Fog
...	...
8475	Fog
8511	Fog
8518	Fog
8537	Fog
8771	Fog

150 rows × 1 columns

Out of **8784** rows there are **150** rows having **foggy** weather condition

data['Weather']: This part of the code extracts the 'Weather' column from the original DataFrame 'data.' It selects only the 'Weather' column.

data['Weather']=='Fog': This part of the code creates a Boolean Series that checks whether each row in the 'Weather' column is equal to the string 'Fog.' It results in a Series of True and False values, indicating which rows have 'Fog' as their weather condition.

data.loc[...]: The .loc indexer is used to select rows from the original DataFrame 'data' based on the condition specified inside the square brackets. In this case, it selects rows where the condition data['Weather']=='Fog' is True.

pd.DataFrame(...): Finally, the selected rows are wrapped in pd.DataFrame(...), which creates a new DataFrame containing only the rows where the 'Weather' column has the value 'Fog.'

So, the resulting DataFrame will contain all the columns from the original DataFrame ('data') but only include rows where the weather condition is 'Fog.' This essentially filters the original data to include only the rows associated with 'Fog' weather conditions.

14. Find all instances when 'Weather is Clear' or 'Visibility is above 40'.

```
data.loc[(data['Weather']=='Clear') | (data['Visibility_km']>40),['Weather','Visibility_km']]
```

	Weather	Visibility_km
0	Clear	25.0
9	Clear	48.3
16	Clear	25.0
17	Mainly Clear	48.3
18	Cloudy	48.3
...	...	...
8774	Mostly Cloudy	48.3
8777	Mainly Clear	48.3
8779	Cloudy	48.3
8780	Mostly Cloudy	48.3
8781	Clear	24.1

3027 rows × 2 columns

data['Weather']=='Clear': This part of the code creates a Boolean Series that checks whether each row in the 'Weather' column of the 'data' DataFrame is equal to the string 'Clear'. It results in a Series of True and False values, indicating which rows have 'Clear' as their weather condition.

data['Visibility_km'] > 40: This part of the code creates another Boolean Series that checks whether each row in the 'Visibility_km' column of the 'data' DataFrame has a value greater than 40. It results in a Series of True and False values, indicating which rows have a visibility greater than 40 kilometers.

(data['Weather']=='Clear') | (data['Visibility_km']>40): This part of the code uses the | operator to combine the two Boolean Series created in steps 1 and 2 using a logical OR operation. This means that it will select rows where either the weather condition is 'Clear' or the visibility is greater than 40 kilometers.

['Weather', 'Visibility_km']: This part of the code within square brackets specifies the columns you want to select from the DataFrame. In this case, you are interested in the 'Weather' and 'Visibility_km' columns.

data.loc[...]: Finally, the .loc indexer is used to select rows and columns based on the conditions and column selections specified above. It will return a DataFrame containing only the rows where either the weather condition is 'Clear' or the visibility is greater than 40 kilometers, and it will include only the 'Weather' and 'Visibility_km' columns for those selected rows.

So, the resulting DataFrame will contain only the rows where either the weather condition is 'Clear' or the visibility is greater than 40 kilometers, and it will show the values in the 'Weather' and 'Visibility_km' columns for those rows.

```
data.loc[(data['Weather']=='Clear')&(data['Rel Hum_%']>50)|(data['Visibility_km']>40),['Weather','Rel Hum_%','Visibility_km']]
```

	Weather	Rel Hum_%	Visibility_km
9	Clear	35	48.3
17	Mainly Clear	42	48.3
18	Cloudy	42	48.3
19	Clear	43	48.3
23	Mainly Clear	45	48.3
...	...	...	...
8774	Mostly Cloudy	92	48.3
8777	Mainly Clear	95	48.3
8779	Cloudy	97	48.3
8780	Mostly Cloudy	98	48.3
8781	Clear	99	24.1

2864 rows × 3 columns

(data['Weather']=='Clear'): This part of the code creates a Boolean Series that checks whether each row in the 'Weather' column of the 'data' DataFrame is equal to the string 'Clear'. It results in a Series of True and False values, indicating which rows have 'Clear' as their weather condition.

(data['Rel Hum_%']>50): This part of the code creates another Boolean Series that checks whether each row in the 'Rel Hum_%' column of the 'data' DataFrame has a value greater than 50. It results in a Series of True and False values, indicating which rows have relative humidity greater than 50%.

(data['Weather']=='Clear') & (data['Rel Hum_%']>50): Here, the & operator is used to combine the Boolean Series created in steps 1 and 2 using a logical AND operation. This means that it will select rows where both the weather condition is 'Clear' and the relative humidity is greater than 50%.

(data['Visibility_km']>40): This part of the code creates yet another Boolean Series that checks whether each row in the 'Visibility_km' column of the 'data' DataFrame has a value

greater than 40 kilometers. It results in a Series of True and False values, indicating which rows have visibility greater than 40 kilometers.

(data['Weather']=='Clear') & (data['Rel Hum_%']>50) | (data['Visibility_km']>40): This part of the code combines the two sets of conditions using the | operator, which represents a logical OR operation. This means it will select rows where either both conditions (Clear weather and relative humidity > 50%) are met or the condition (Visibility > 40 km) is met.

['Weather', 'Rel Hum_%', 'Visibility_km']: This part of the code within square brackets specifies the columns you want to select from the DataFrame. In this case, you are interested in the 'Weather,' 'Rel Hum_%,' and 'Visibility_km' columns.

data.loc[...]: Finally, the .loc indexer is used to select rows and columns based on the combined conditions and column selections specified above. It will return a DataFrame containing rows where either the weather condition is 'Clear' and the relative humidity is greater than 50% or the visibility is greater than 40 kilometers. For these selected rows, it will show the values in the 'Weather,' 'Rel Hum_%,' and 'Visibility_km' columns.

So, the resulting DataFrame will contain rows that meet the combined conditions, and it will display the specified columns for those rows.